

Bab 1. Bahasa Pemrograman Java

OBJEKTIF :

1. Mahasiswa mampu belajar bahasa pemrograman Java seperti mengetahui variabel, attribute, dan konstanta.
 2. Mahasiswa mampu menggunakan tipe data dengan baik.
 3. Mahasiswa mampu menggunakan operator Java dengan baik.
 4. Mahasiswa mampu memahami tentang *package* dan *library* dalam pemrograman java.
 5. Mahasiswa mampu melakukan *input* dan *output* pada Java.
 6. Mahasiswa mampu memahami algoritma, diagram uml, *pseudocode*, dan *flowchart*.
-

1.1 Dasar - Dasar Program Java

1.1.1 Definisi Variabel

Variabel adalah tempat (lokasi memori) di dalam memori utama dimana kita bisa menyimpan data atau nilai kita. Sebuah variabel harus dideklarasikan sebelum kita bisa menyimpan nilai di dalamnya. Saat membuat variabel di Java, kita perlu menentukan tipe data yang ingin kita gunakan. Tipe data variabel tidak akan bisa dirubah ketika sudah dideklarasikan. Sebagai contoh, jika kita sudah membuat variabel dengan tipe data `double`, maka variabel dengan tipe data tersebut tidak bisa dirubah menjadi variabel dengan tipe data `string` atau yang lainnya.

Membuat Variabel

Untuk membuat variabel dalam java kita harus melakukan 3 hal berikut.

1. Menentukan tipe data yang ingin kita buat dalam variabel seperti `int`, `float`, `string`, dll. Tipe data mempengaruhi jenis data yang dapat ditempatkan dalam variabel.
2. Memberikan nama variabel yang sesuai dengan aturan *syntax* variabel.
3. Memberikan nilai variabel. Nilai variabel adalah data yang ditempatkan dalam variabel.

Berikut *syntax* dasar dalam membuat variabel pada java.

```
<tipe_data> <nama_variabel>;
```

Kita juga harus memperhatikan aturan-aturan dalam pembuatan variabel di java. Berikut aturan dalam pembuatan variabel dalam java.

1. Nama variabel harus dimulai dengan huruf, tetapi dapat mengandung angka, garis bawah, atau garis bawah di akhir. Contoh `String namaMahasiswa;`, `float gaji_;`, `int _nilai;`.
2. Nama variabel **tidak boleh** berisi spasi atau karakter khusus seperti `!`, `@`, `#`, dan lainnya. Contoh `int kelas mahasiswa;`, `float @gaji;`.
3. Nama variabel **tidak boleh** sama dengan nama bahasa pemrograman Java atau nama *class*. Contoh `int class;`, `double public;`.
4. Tipe data variabel harus dideklarasikan sebelum variabel dapat digunakan. Contoh `String nama_mahasiswa;`, `int umur;`.
5. Nama variabel harus diperhatikan untuk *case-sensitivity*. Perbedaan huruf besar dan kecil akan membuat dua nama variabel yang berbeda. Contoh `int umurkamu;`, `int umurKamu;` kedua variabel ini memiliki nilai yang berbeda.

6. Nama variabel disarankan menggunakan *camelCase* yang artinya semua suku kata menyatu dan terdapat huruf kapital untuk memisahkannya. Contoh `String namaKamu`, `int kelasKamu`, `int nilaiKamu`.

Pemanggilan Variabel

Pada java, kita dapat memanggil variabel dengan menggunakan nama variabelnya. Berikut merupakan contoh pemanggilan variabel dalam Java.

```
// Membuat variabel umur
int umur = 20;

// Memanggil variabel umur
System.out.println("Umur saya adalah " + umur + " tahun");
```

Output Program

```
Umur saya adalah 20 tahun
```

Kita membuat variabel umur bernama `umur` dan memberikan nilai `20`. Kemudian, kita memanggil variabel tersebut dengan menuliskan `umur` di dalam *statement* `System.out.println()`. Kata-kata yang ada di dalam tanda kutip dan `+` akan digabungkan dengan nilai variabel `umur` sehingga menghasilkan *output* "Umur saya adalah 20 tahun".

Penggunaan Variabel

Berikut ini adalah beberapa penggunaan variabel dalam Java.

1. Penyimpanan data: Variabel digunakan untuk menyimpan berbagai jenis data, seperti angka, teks, atau *object*.
2. Pengulangan: Variabel dapat digunakan untuk mengontrol perulangan seperti *for loop* dan *while loop*.
3. Kondisi: Variabel dapat digunakan untuk membuat kondisi seperti *if-else statements* dan *switch statements*.
4. Menyimpan hasil dari operasi: Variabel dapat digunakan untuk menyimpan hasil dari operasi matematika atau logika.
5. Mengirim data dari satu *method* ke *method* lain: Variabel dapat digunakan untuk mengirim data dari satu *method* ke *method* lain, sehingga memudahkan komunikasi antar *method*.
6. Pengaturan *default value*: Variabel dapat diberikan nilai *default* sebelum diinisialisasi, sehingga memudahkan proses inisialisasi data.

1.1.2 Definisi Attribute

Atribut (*attribute*) adalah variabel pada *class* yang menyatakan properti atau karakteristik atau ciri dari suatu *object*. *Attribute* membuat *object* memiliki nilai yang berbeda dari *object* lain meskipun mereka merupakan instansi dari *class* yang sama. *Attribute* pada *object* menentukan bagaimana *object* tersebut akan terlihat dan berperilaku. Sebagai contoh, misal terdapat sebuah *class* dengan nama *class*-nya adalah `Car` maka *attribute*-nya dapat mencakup warna, tipe mesin, merek, tahun produksi, dan lainnya. Setiap *object* yang dibuat dari *class* `Car` akan memiliki set *attribute* yang berbeda.

Perbedaan *attribute* dan variabel

Attribute dan variabel adalah konsep yang berbeda dalam pemrograman Java. Meskipun setiap *attribute* adalah variabel, namun variabel belum tentu sebuah *attribute*. Pengertian *attribute* merujuk pada variabel pada *class* yang menjadi ciri atau karakteristik dari suatu *object*. Sedangkan variabel yang terdapat di dalam fungsi atau *method* tidak disebut sebagai *attribute*. Sebagai contoh, *attribute* tidak dapat dipanggil secara langsung, sementara variabel dapat. *Attribute* harus diakses melalui *object* yang memilikinya, sedangkan variabel dapat dipanggil langsung dari *method* atau blok kode.

Membuat *Attribute*

Berikut *syntax* untuk membuat *attribute* dalam java.

```
<modifier> <tipe_data> <nama_atribut>;
```

atau

```
<modifier> <tipe_data> <nama_atribut> = <nilai>;
```

- `modifier` adalah kata kunci opsional seperti `public`, `private`, `protected`, atau `default` yang menentukan aksesibilitas *attribute*.
- `tipe_data` adalah tipe data dari *attribute*, seperti `int`, `float`, `String`, dll.
- `nama_atribut` adalah nama dari *attribute*.
- `nilai` adalah penugasan nilai opsional pada *attribute*.

Berikut merupakan contoh penulisan *attribute*.

```
public int id;  
private String nama;  
protected double gaji = 10000.0;
```

Pemanggilan *Attribute*

Untuk memanggil *atributte* dari *object* lain adalah dengan membuat *object* dari *class* yang memiliki *attribute* yang ingin dipanggil, lalu gunakan operator titik `.` untuk mengakses *atribut* tersebut. Berikut merupakan contoh pemanggilan *attribute* diluar *object*.

Definisi Class (*Mahasiswa.java*)

```
class Mahasiswa  
{  
    public int npm = 1122312;  
    public String nama = "andi";  
    public int umur = 20 ;  
}
```

Program (*Main.java*)

```
public class Main  
{  
    public static void main(String[] args)  
    {  
  
        // Membuat object dari kelas Mahasiswa
```

```

Mahasiswa mhs = new Mahasiswa();

// Mengakses atribut id dari object mhs
System.out.println("NPM mahasiswa: " + mhs.npm);

// Mengakses atribut nama dari object mhs
System.out.println("Nama mahasiswa: " + mhs.nama);

// Mengakses atribut umur dari object mhs
System.out.println("Umur mahasiswa: " + mhs.umur);
}
}

```

Output Program (Main.java)

```

NPM mahasiswa: 1122312
Nama mahasiswa: andi
Umur mahasiswa: 20

```

Penggunaan *Attribute*

Penggunaan *attribute* dalam Java antara lain:

1. Menyimpan informasi: *Attribute* digunakan untuk menyimpan informasi atau data yang relevan untuk *object*, misalnya nama, alamat, gaji, dll.
2. Mempermudah akses data: Dengan menyimpan data pada *attribute*, kita dapat mengakses data tersebut dengan mudah tanpa harus membuat *method* baru untuk setiap data.
3. Mempermudah pemeliharaan kode: Dengan menyimpan data pada *attribute*, kita dapat memelihara kode dengan lebih mudah dan terstruktur.
4. Mempermudah implementasi *inheritance*: *Attribute* juga dapat digunakan pada *inheritance* atau pewarisan kelas, sehingga mempermudah implementasi.

1.1.2 Definisi Konstanta

Konstanta adalah sebuah nilai yang tidak dapat diubah setelah dideklarasikan. Dalam pemrograman Java, konstanta dideklarasikan dengan menambahkan *keyword* `final` sebelum tipe data, seperti `final int x = 10;` yang menandakan bahwa nilai dari `x` selalu `10` sepanjang program berjalan. Konstanta digunakan untuk membatasi nilai dari suatu variabel agar tidak dapat diubah sehingga memberikan keamanan dan mempermudah pemeliharaan kode program.

Membuat Konstanta

Untuk membuat konstanta dalam Java, berikut ini adalah langkah-langkahnya:

1. Tambahkan *keyword* `final` sebelum tipe data pada saat deklarasi variabel.
2. Berikan nilai pada variabel saat deklarasi.
3. Gunakan nama variabel dengan huruf besar dan menggunakan *underscore* untuk memisahkan kata.

Berikut merupakan contoh penulisan sebuah konstanta.

```

final int NILAI_AKHIR = 100;

```

Pemanggilan Konstanta

Untuk pemanggilan konstanta dalam java sama seperti variabel , Berikut adalah contoh pemanggilan konstanta dalam Java:

```
public class nilai
{
    public static final double PI = 3.14;

    public static void main(String[] args)
    {
        System.out.println("Nilai PI adalah: " + PI);
    }
}
```

Penggunaan Konstanta

Penggunaan konstanta dalam Java dapat membantu membuat program lebih terstruktur dan mudah dipelihara. Berikut adalah beberapa contoh penggunaan konstanta dalam Java.

1. Nilai tetap yang sering digunakan: Dalam beberapa kasus, ada nilai tetap yang sering digunakan dalam program, seperti pi, gravitasi bumi, dan sebagainya. Dengan menggunakan konstanta, kita dapat menentukan nilai tersebut sekali dan menggunakannya berulang-ulang di program tanpa harus menulis ulang nilainya setiap kali.
2. Konfigurasi program: Konstanta dapat digunakan untuk menyimpan konfigurasi program seperti batas maksimal, batas minimum, dan sebagainya. Ini membuat program lebih mudah dipelihara karena kita hanya perlu mengubah nilai konstanta sekali jika diperlukan, dan perubahan akan terjadi di seluruh program.
3. Mempermudah pemahaman kode: Dengan menggunakan konstanta, kita dapat memberikan nama yang lebih memahami untuk nilai tetap, sehingga mempermudah pemahaman kode oleh orang lain atau oleh diri sendiri pada waktu yang akan datang.

1.2 Tipe data

Dalam pemrograman Java, tipe data adalah kategori yang membedakan antara jenis informasi yang dapat ditampung oleh variabel dalam suatu program. Tipe data dalam Java dapat dikelompokkan secara umum ke dalam dua kategori. Tipe data *reference* dan tipe data primitif.

1.2.1 Tipe Data Primitif

Tipe data primitif adalah tipe data dasar dalam Java yang tidak berorientasi *object*. Tipe data primitif memiliki tipe data dan nilai yang ditentukan dan tidak dapat diubah. Tipe data primitif hanya mampu menyimpan satu nilai tiap satu variabelnya yang sudah didefinisikan oleh Java. Berikut ini beberapa tipe data primitif dalam java.

Integer

Tipe data integer dalam Java adalah tipe data primitif yang digunakan untuk menampung angka bulat. Tipe data integer sering digunakan dalam pemrograman untuk menyimpan informasi seperti jumlah item, ukuran, ID, dll. Dalam Java, angka bulat harus ditulis dengan tanda tanpa desimal, misalnya 100, -5, dll. Tipe data integer juga dapat dilakukan operasi aritmatika seperti penjumlahan, pengurangan, perkalian, dan pembagian. Tipe data integer meliputi empat ukuran yang berbeda, yaitu:

1. `byte`: Tipe data ini memiliki ukuran 1 *byte* dan digunakan untuk menampung angka bulat yang berukuran kecil. Nilai minimum yang dapat ditampung adalah -128 dan nilai maksimum adalah 127.
2. `short`: Tipe data ini memiliki ukuran 2 *byte* dan digunakan untuk menampung angka bulat yang sedikit lebih besar daripada *byte*. Nilai minimum yang dapat ditampung adalah -32768 dan nilai maksimum adalah 32767.
3. `int`: Tipe data ini memiliki ukuran 4 *byte* dan digunakan untuk menampung angka bulat yang lebih besar lagi. Nilai minimum yang dapat ditampung adalah -2147483648 dan nilai maksimum adalah 2147483647.
4. `long`: Tipe data ini memiliki ukuran 8 *byte* dan digunakan untuk menampung angka bulat yang sangat besar. Nilai minimum yang dapat ditampung adalah -9223372036854775808 dan nilai maksimum adalah 9223372036854775807.

Berikut merupakan contoh penggunaan tipe data integer.

Program (Main.java)

```
public class Main
{
    public static void main(String[] args)
    {
        int totalMahasiswa = 1023;

        System.out.println(totalMahasiswa);
    }
}
```

Output Program (Main.java)

```
1023
```

Pada contoh diatas kita mendeklarasikan variabel `totalMahasiswa` dengan tipe data `int` lalu kita menugaskan variabel tersebut dengan nilai `totalMahasiswa=1023`.

Floating Point

Tipe data *floating point* adalah tipe data primitif dalam Java yang digunakan untuk menampung angka pecahan (desimal). Angka pecahan harus ditulis dengan tanda desimal, misalnya 3.14, -0.5, 1.3, dll. Tipe data *floating point* sangat cocok untuk menyimpan informasi seperti ukuran, jarak, suhu, dll. Tipe data *floating point* juga dapat dilakukan operasi aritmatika seperti penjumlahan, pengurangan, perkalian, dan pembagian. Berikut dua tipe data floating point dalam Java.

1. `float`: Tipe data ini memiliki ukuran 4 *byte* dan digunakan untuk menampung angka pecahan dengan presisi sedang. Nilai minimum yang dapat ditampung adalah 1.4×10^{-45} dan nilai maksimum adalah 3.4028235×10^{38} .
2. `double`: Tipe data ini memiliki ukuran 8 *byte* dan digunakan untuk menampung angka pecahan dengan presisi tinggi. Nilai minimum yang dapat ditampung adalah 4.9×10^{-324} dan nilai maksimum adalah $1.7976931348623157 \times 10^{308}$.

Berikut merupakan contoh penggunaan tipe data *floating point*.

Program (Main.java)

```
public class Main
{
    public static void main(String[] args)
    {
        float myNum1 = 5.75f;
        double myNum2 = 5.75;

        System.out.println(myNum1);
        System.out.println(myNum2);
    }
}
```

Output Program (Main.java)

```
5.75
5.75
```

Pada contoh diatas kita mendeklarasikan variabel `myNum1` dan `myNum2` dengan tipe data `float` dan `double` lalu kita menugaskan variabel tersebut dengan nilai yang sama yaitu `5.75`. Perhatikan cara mengisi variabel `myNum1`, terdapat akhiran `f`, yakni `5.75f`. Ini diperlukan karena secara *default* semua angka pecahan di Java dianggap sebagai `double`. Akhiran `f` untuk proses *input* tipe data `float` ini harus ditulis, jika tidak akan terjadi error.

Boolean

Tipe data `boolean` digunakan untuk menyimpan nilai boolean, yaitu `true` atau `false`. Nilai boolean digunakan untuk melakukan pengambilan keputusan dalam program, misalnya memeriksa apakah suatu kondisi terpenuhi atau tidak. Berikut contoh program menggunakan tipe data `boolean`.

Program (StatusVaksin.java)

```
public class StatusVaksin
{
    public static void main(String[] args)
    {
        boolean vaksinPertama, vaksinkedua;
        vaksinPertama = true;
        System.out.println(vaksinPertama);
        vaksinkedua = false;
        System.out.println(vaksinkedua);
    }
}
```

Output Program (StatusVaksin.java)

```
true
false
```

Variabel yang bertipe data `boolean` berguna untuk mengevaluasi kondisi yang bersifat benar (`true`) atau salah (`false`). Variabel bertipe `boolean` akan sangat berguna saat kita memasuki materi Struktur Keputusan.

Char

Tipe data `char` adalah tipe data yang digunakan untuk menyimpan karakter *Unicode*. *Unicode* adalah standar yang digunakan untuk menentukan representasi karakter dalam komputer, dan mencakup berbagai jenis karakter dari berbagai bahasa.

Sebuah karakter *Unicode* dalam Java dapat didefinisikan dengan menggunakan tanda petik tunggal (`'`). Berikut merupakan contoh penggunaan tipe data `char`.

```
char golonganDarah;  
golonganDarah = 'A';
```

Pada contoh diatas kita mendeklarasikan variabel `golonganDarah` dengan tipe data `char` lalu kita menugaskan variabel tersebut dengan nilai `A`.

1.2.2 Tipe data *reference*

Tipe data *reference* digunakan dalam OOP (*Object Oriented Programming*) untuk mereferensikan *object* atau *class* tertentu, seperti *String*, *Interface*, dan *Array*. Tipe data *reference* mampu menyimpan banyak atau lebih dari satu nilai dan didefinisikan sendiri oleh penggunaanya.

String

Tipe data `String` adalah tipe data *reference* yang digunakan untuk menyimpan sekumpulan karakter. Dalam Java, tipe data `String` tidak dapat didefinisikan sebagai tipe data primitif, namun harus didefinisikan sebagai *object*.

Contoh penggunaan tipe data `String` dalam Java adalah sebagai berikut.

Program (Main.java)

```
public class Main  
{  
    public static void main(String[] args)  
    {  
        String myString = "Hello world";  
  
        System.out.println("My String: " + myString);  
    }  
}
```

Output Program (Main.java)

```
My String: Hello world
```

Pada pemrograman menggunakan Java, dapat dilakukan penggabungan atau konkatensi dua *string* dengan menggunakan operator tambah `+`. Berikut contoh penggabungan dua *string* / konkatensi *string*.

Program (Main.java)

```
public class Main
{
    public static void main(String[] args)
    {
        String string1 = "Hello";
        String string2 = "World";

        System.out.println("konkatenasi String: " + string1 + " " + string2);
    }
}
```

Output Program (Main.java)

```
konkatenasi String: Hello World
```

Dalam contoh diatas, kita menyimpan dua *string* pada variabel `string1` dan `string2`, dan menggabungkan kedua *string* tersebut menjadi satu *string* baru dengan menggunakan operator tambah `+`. Hasil penggabungan dari kedua *string* tersebut akan disimpan pada variabel baru dan ditampilkan pada *console*.

1.2.3 Konversi Tipe Data

Konversi tipe data dalam Java adalah proses mengubah tipe data suatu variabel menjadi tipe data lain. Misalkan, ketika kita ingin melakukan perubahan tipe data `int` pada suatu variabel menjadi tipe data `double` maupun sebaliknya. Terdapat dua jenis konversi tipe data, yaitu konversi tipe data secara otomatis dan konversi tipe data manual.

Konversi Otomatis

Konversi tipe data secara otomatis terjadi ketika Java melakukan penentuan tipe data yang sesuai untuk sebuah nilai atau variabel. Contoh, ketika sebuah variabel integer diassign ke sebuah variabel *double*, Java akan melakukan konversi tipe data secara otomatis dari integer ke *double*, seperti contoh berikut.

Program (Main.java)

```
public class Main
{
    public static void main(String[] args)
    {
        int myInt = 20;
        double myDouble = myInt;

        System.out.println("My Integer: " + myInt);
        System.out.println("My Double: " + myDouble);
    }
}
```

Output Program (Main.java)

```
My Integer: 20  
My Double: 20.0
```

Dalam contoh ini, sebuah variabel integer `myInt` memiliki nilai 20, dan kita meng-asign-nya ke sebuah variabel *double* `myDouble`. Dalam hal ini, Java akan melakukan konversi tipe data secara otomatis dari integer ke *double*, dan hasil dari konversi tipe data ini akan disimpan pada variabel *double* `myDouble`.

Konversi Manual

Konversi tipe data secara manual adalah proses mengubah tipe data dari satu tipe data ke tipe data lain secara eksplisit. Hal ini dilakukan untuk memastikan tipe data yang diinginkan, meskipun konversi tipe data secara otomatis sudah memungkinkan.

Berikut ini adalah contoh konversi tipe data secara manual dalam Java.

Program (Main.java)

```
public class Main  
{  
    public static void main(String[] args)  
    {  
        double myDouble = 8.50;  
        int myInt = (int) myDouble;  
  
        System.out.println("My Double: " + myDouble);  
        System.out.println("My Integer: " + myInt);  
    }  
}
```

Output Program (Main.java)

```
My Double: 8.5  
My Integer: 8
```

Dalam hal ini, setelah kita mengubah tipe data *double* `myDouble` ke tipe data integer `myInt`, hasil konversi tipe data secara manual akan memotong bagian pecahan (0.50) dan hanya menyimpan bagian bulat (8) dari nilai asal.

1.3 Operator Java

Java menyediakan beragam operator. Operator menggabungkan konstanta, variabel, dan sub-ekspresi untuk membentuk ekspresi. Sebagian besar operator dalam Java bertindak seperti C/C++, tetapi terdapat beberapa perbedaan yang ada di java. Berikut operator yang ada dalam java.

1.3.1 Operator Aritmatika dan *Math class*

Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk melakukan operasi matematika seperti penjumlahan, pengurangan, perkalian, pembagian, dan lain-lain. Berikut ini adalah beberapa operator aritmatika dalam Java.

1. Penjumlahan (`+`): digunakan untuk menjumlahkan dua angka. Berikut contoh program penjumlahan dalam operator aritmatika.

Program (Penjumlahan.java)

```
public class Penjumlahan
{
    public static void main(String args[])
    {
        int x = 5;
        int y = 10;
        int z = x + y;

        System.out.println("Hasil penjumlahan: " + z);
    }
}
```

Output Program (Penjumlahan.java)

```
Hasil penjumlahan: 15
```

2. Pengurangan (`-`): digunakan untuk mengurangi dua angka. Berikut contoh program pengurangan dalam operator aritmatika.

Program (Pengurangan.java)

```
public class Pengurangan
{
    public static void main(String args[])
    {
        int x = 10;
        int y = 5;
        int z = x - y;

        System.out.println("Hasil pengurangan: " + z);
    }
}
```

Output Program (Pengurangan.java)

```
Hasil pengurangan: 5
```

3. Perkalian (`*`): digunakan untuk mengalikan dua angka. Berikut contoh program perkalian dalam operator aritmatika.

Program (Perkalian.java)

```
public class Perkalian
{
    public static void main(String args[])
    {
        int x = 5;
        int y = 10;
        int z = x * y;

        System.out.println("Hasil perkalian: " + z)
    }
}
```

Output Program (Perkalian.java)

Hasil perkalian: 50

4. Pembagian (/): digunakan untuk membagi dua angka. Berikut contoh program pembagian dalam operator aritmatika.

Program (Pembagian.java)

```
public class Pembagian
{
    public static void main(String args[])
    {
        int x = 40;
        int y = 10;
        int z = x / y;

        System.out.println("Hasil pembagian: " + z);
    }
}
```

Output Program (Pembagian.java)

Hasil pembagian: 4

5. Modulus (%): digunakan untuk mencari sisa bagi dari pembagian dua angka. Berikut contoh program modulus dalam operator aritmatika.

Program (Modulus.java)

6.

```
public class Modulus
{
    public static void main(String args[])
    {
        int x = 5;
        int y = 4;
        int z = x % y;

        System.out.println("Hasil sisa bagi: " + z);
    }
}
```

Output Program (Modulus.java)

Hasil sisa bagi: 1

Math Class

Saat kita membuat program Java seringkali kita menggunakan operator-operator aritmatika yang lebih kompleks. Seperti akar kuadrat, pemangkatan, logaritma dan lain sebagainya. Dalam Java sendiri tidak ada simbol untuk operasi-operasi tersebut. Untuk melakukan operasi-operasi tersebut kita harus menggunakan *method* dari class `Math`. Class `Math` sendiri sudah tersedia dalam Java, kita bisa langsung menggunakannya untuk operasi-operasi yang kita inginkan. Berikut merupakan *method* yang ada dalam class `Math`.

Method	Menghitung
<code>Math.sqrt(x)</code>	Akar kuadrat dari x
<code>Math.pow(x, y)</code>	x^y
<code>Math.sin(x)</code>	Sinus dari x (x dalam radian)
<code>Math.cos(x)</code>	Cosinus dari x (x dalam radian)
<code>Math.exp(x)</code>	e^x
<code>Math.log(x)</code>	Logaritma natural dari x ($\ln(x)$)
<code>Math.log10(x)</code>	Logaritma basis 10 dari x ($\log_{10}(x)$)
<code>Math.round(x)</code>	Pembulatan x ke integer terdekat
<code>Math.abs(x)</code>	Nilai mutlak dari x ($ x $)
<code>Math.max(x, y)</code>	Maksimum dari x dan y
<code>Math.min(x, y)</code>	Minimum dari x dan y

Berikut contoh program class `math` dalam Java.

Program (MathClass.java)

```
public class MathClass
{
    public static void main(String[] args)
    {
        //Menggunakan method abs untuk menghitung nilai absolut
        System.out.println("Nilai absolut dari -26 adalah : " + Math.abs(-26));

        //Menggunakan method max untuk mencari nilai maksimum dari 2 bilangan
        System.out.println("Nilai maksimum dari 23 dan 56 adalah : " + Math.max(23,
56));

        //Menggunakan method min untuk mencari nilai minimum dari 2 bilangan
        System.out.println("Nilai minimum dari 23 dan 56 adalah : " + Math.min(23,
56));

        //Menggunakan method pow untuk menghitung bilangan pangkat
```

```

        System.out.println("Hasil dari 3 pangkat 4 adalah : " + Math.pow(3, 4));

        //Menggunakan method sqrt untuk menghitung akar kuadrat
        System.out.println("Akar kuadrat dari 9 adalah : " + Math.sqrt(9));
    }
}

```

Output Program (MathClass.java)

```

Nilai absolut dari -26 adalah : 26
Nilai maksimum dari 23 dan 56 adalah : 56
Nilai minimum dari 23 dan 56 adalah : 23
Hasil dari 3 pangkat 4 adalah : 81.0
Akar kuadrat dari 9 adalah : 3.0

```

1.3.2 Operator Penugasan

Operator penugasan adalah operator yang digunakan untuk menetapkan nilai ke suatu variabel. Operator ini memiliki *syntax* seperti : `variabel = nilai`. Operator penugasan ini biasanya digunakan sebagai cara untuk mengisi variabel dengan nilai tertentu. Ada beberapa jenis operator penugasan dalam Java, di antaranya:

1. Operator penugasan sederhana (=): menugaskan nilai ke suatu variabel, misalnya `int x = 10;`
2. Operator penugasan jumlah (+=): menambahkan nilai ke suatu variabel, misalnya `int x += 5;`
3. Operator penugasan pengurangan (-=): mengurangi nilai dari suatu variabel, misalnya `int x -= 5;`
4. Operator penugasan perkalian (*=): mengalikan nilai dari suatu variabel, misalnya `int x *= 5;`
5. Operator penugasan pembagian (/=): membagi nilai dari suatu variabel, misalnya `int x /= 5;`
6. Operator penugasan sisa bagi (%=): menghitung sisa bagi dari suatu variabel, misalnya `int x %= 5;`

Berikut contoh program dalam mengaplikasikan operator penugasan.

Program (OperatorPenugasan.java)

```

public class OperatorPenugasan
{
    public static void main(String[] args)
    {
        int num1 = 10;
        int num2 = 20;

        num2 += 100;
        System.out.println("Hasil dari num2 += 100 adalah: " + num2);

        num2 -= 5;
        System.out.println("Hasil dari num2 -= 5 adalah: " + num2);

        num2 *= num1;
        System.out.println("Hasil dari num2 *= num1 adalah: " + num2);
    }
}

```

```

        num2 /= num1;
        System.out.println("Hasil dari num2 /= num1 adalah: " + num2);

        num2 %= num1;
        System.out.println("Hasil dari num2 %= num1 adalah: " + num2);
    }
}

```

Output Program (OperatorPenugasan.java)

```

Hasil dari num2 += 100 adalah: 120
Hasil dari num2 -= 5 adalah: 115
Hasil dari num2 *= num1 adalah: 1150
Hasil dari num2 /= num1 adalah: 115
Hasil dari num2 %= num1 adalah: 5

```

1.3.3 Operator Pembandingan

Operator pembandingan adalah operator yang digunakan untuk membandingkan dua buah nilai, dan hasil dari operasi pembandingan ini adalah *boolean* (*true* atau *false*). Berikut adalah macam-macam operator pembandingan dalam Java.

Operator	Keterangan
==	sama dengan
!=	tidak sama dengan
>	lebih dari
<	kurang dari
>=	lebih dari sama dengan
<=	kurang dari sama dengan

Berikut merupakan contoh program dalam mengaplikasikan operator pembandingan.

Program (OperatorPembandingan.java)

```

public class OperatorPembandingan
{
    public static void main(String[] args)
    {
        int a = 10;
        int b = 20;

        System.out.println(a == b); // hasilnya false
        System.out.println(a != b); // hasilnya true
        System.out.println(a < b); // hasilnya true
        System.out.println(a <= b); // hasilnya true
        System.out.println(a > b); // hasilnya false
        System.out.println(a >= b); // hasilnya false
    }
}

```

Output Program (OperatorPembanding.java)

```
false
true
true
true
false
false
```

1.3.4 Operator Logika

Operator logika dipakai untuk menghasilkan nilai *boolean true* atau *false* dari 2 kondisi atau lebih. Berikut adalah macam-macam operator logika dalam Java.

Operator	Keterangan	Penjelasan
&&	And	Menghasilkan true jika kedua operand true
	Or	Menghasilkan true jika salah satu operand true
!	Not	Menghasilkan true jika operand false

Berikut contoh program dalam mengaplikasikan operator logika.

Program (OperatorLogika.java)

```
public class OperatorLogika
{
    public static void main(String args[])
    {
        boolean a = true;
        boolean b = false;
        boolean hasil;

        hasil = a && a;
        System.out.println("Hasil dari a && a : " + hasil );

        hasil = a && b;
        System.out.println("Hasil dari a && b : " + hasil );

        hasil = a || b;
        System.out.println("Hasil dari a || b : " + hasil );

        hasil = b || b;
        System.out.println("Hasil dari b || b : " + hasil );

        hasil = !a;
        System.out.println("Hasil dari !a : " + hasil );

        hasil = !b;
        System.out.println("Hasil dari !b : " + hasil );
    }
}
```


Output Program (OperatorLogika.java)

```
Hasil dari a && a : true
Hasil dari a && b : false
Hasil dari a || b : true
Hasil dari b || b : false
Hasil dari !a : false
Hasil dari !b : true
```

1.3.5 Operator Bitwise

Operator *Bitwise* adalah operator khusus untuk menangani operasi logika bilangan biner dalam bentuk *bit*. Bilangan biner sendiri merupakan jenis bilangan yang hanya terdiri dari 2 jenis angka, yakni 0 dan 1. Jika nilai asal yang dipakai bukan bilangan biner, akan dikonversi secara otomatis oleh *compiler* Java menjadi bilangan biner. Berikut contoh untuk mengkonversi bilangan integer ke bilangan biner.

$$1101 = (1x2^3) + (1x2^2) + (0x2^1) + (1x2^0) = 8 + 4 + 0 + 1 = 13$$

$$1001 = (1x2^3) + (0x2^2) + (0x2^1) + (1x2^0) = 8 + 0 + 0 + 1 = 9$$

Dalam penerapannya, operator *bitwise* tidak terlalu sering dipakai. Selain itu operator ini cukup rumit dan harus memiliki pemahaman tentang sistem bilangan biner. Bahasa Java mendukung 6 jenis operator *bitwise*.

1. *Bitwise* AND (`&`) : Operator ini menghasilkan 1 **hanya jika kedua *bit*** dari operand-nya bernilai 1.

Program (OperatorBitwise.java)

```
public class OperatorBitwise
{
    public static void main(String args[])
    {
        int a = 60; // 60 = 00111100
        int b = 13; // 13 = 00001101

        System.out.println(a&b);
    }
}
```

Output Program (OperatorBitwise.java)

```
12
```

Hasil dari *output* program tersebut adalah 12.

$$00001100 = (0x2^7) + (0x2^6) + (0x2^5) + (0x2^4) + (1x2^3) + (1x2^2) + (0x2^1) + (0x2^0) = 12$$

2. *Bitwise* OR (`|`) : Operator ini menghasilkan 1 **jika setidaknya salah satu *bit*** dari operand-nya bernilai 1.

Program (OperatorBitwise.java)

```
public class OperatorBitwise
{
    public static void main(String args[])
    {
        int a = 60; // 60 = 00111100
        int b = 13; // 13 = 00001101

        System.out.println(a|b);
    }
}
```

Output Program (OperatorBitwise.java)

61

Hasil dari *output* program tersebut adalah 61.

$$00111101 = (0x2^7) + (0x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (0x2^1) + (1x2^0) = 61$$

3. *Bitwise XOR* (\wedge) : Operator ini menghasilkan 1 **jika hanya salah satu bit** dari operand-nya bernilai 1.

Program (OperatorBitwise.java)

```
public class OperatorBitwise
{
    public static void main(String args[])
    {
        int a = 60; // 60 = 00111100
        int b = 13; // 13 = 00001101

        System.out.println(a^b);
    }
}
```

Output Program (OperatorBitwise.java)

49

Hasil dari *output* program tersebut adalah 49.

$$00110001 = (0x2^7) + (0x2^6) + (1x2^5) + (1x2^4) + (0x2^3) + (0x2^2) + (0x2^1) + (1x2^0) = 49$$

4. *Bitwise NOT* (\sim) : Atau biasa disebut dengan Operator Komplemen, Operator ini **mengubah nilai bit** dari suatu operand, dengan cara mengubah 0 menjadi 1 dan sebaliknya.

Program (OperatorBitwise.java)

```
public class OperatorBitwise
{
    public static void main(String args[])
    {

        int a = 60; // 60 = 00111100
        int b = ~a;

        System.out.println("Nilai awal a: " + a);
        System.out.println("Hasil bitwise NOT a: " + b);
    }
}
```

Output Program (OperatorBitwise.java)

```
Nilai awal a: 60
Hasil bitwise NOT a: -61
```

Perhatikan nilai awal `a` dalam bilangan biner adalah :

```
60 = 00111100
```

Dengan mengimplementasikan operator *Bitwise* NOT, maka hasilnya akan menjadi :

```
00111100 -> 11000011
```

Ini adalah komplemen pertama dari angka 60. Dan karena *bit* pertama (paling kiri) adalah 1, maka dalam biner itu berarti tanda negatif untuk angka yang disimpan. Pada Java, bilangan yang disimpan adalah komplemen kedua + 1, maka perlu dicari komplemen keduanya dan mengubah bilangan biner tersebut menjadi suatu angka.

```
11000011 -> 00111100 + 1 = 00111101
```

Maka diperoleh bilangan biner 00111101, selanjutnya akan diubah menjadi suatu angka.

$$00111101 = (0x2^7) + (0x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (0x2^1) + (1x2^0) = 61$$

Karena diawal hasil Operator Komplemen disimpan sebagai bilangan negatif, maka *output* dari program menjadi -61.

5. *Left Shift* (`<<`) : Operator ini **memindahkan nilai *bit*** suatu operand **ke kiri sebanyak jumlah bit yang ditentukan** oleh operand kedua.

Program (OperatorBitwise.java)

```
public class OperatorBitwise
{
    public static void main(String args[])
    {
        int a = 60; // 60 = 00111100
        int b = a << 1; // a << 1 = 01111000

        System.out.println(b);
    }
}
```

Output Program (OperatorBitwise.java)

120

Hasil dari *output* program tersebut adalah 120.

$$01111000 = (0x2^7) + (1x2^6) + (1x2^5) + (1x2^4) + (1x2^3) + (0x2^2) + (0x2^1) + (0x2^0) = 120$$

6. *Right Shift* (`>>`) : Operator ini **memindahkan nilai *bit*** suatu operand **ke kanan sebanyak jumlah bit yang ditentukan** oleh operand kedua.

Program (OperatorBitwise.java)

```
public class OperatorBitwise
{
    public static void main(String args[])
    {
        int a = 60; // 60 = 00111100
        int b = a >> 1; // a >> 1 = 00011110

        System.out.println(b);
    }
}
```

Output Program (OperatorBitwise.java)

30

Hasil dari *output* program tersebut adalah 30.

$$00011110 = (0x2^7) + (0x2^6) + (0x2^5) + (1x2^4) + (1x2^3) + (1x2^2) + (1x2^1) + (0x2^0) = 30$$

1.4 Package and Library

1.4.1 Package

Package merupakan sebuah cara untuk mengelompokkan *class*. *Package* bertujuan untuk menghindari terjadinya bentrok yang disebabkan oleh nama *class* yang sama dan juga untuk mempermudah proses penggunaan *class* yang dibutuhkan. *Package* dapat diibaratkan sebagai folder yang berisi *class-class* yang sesuai dengan kegunaannya. Satu *package* tidak boleh berisi dua *class* dengan nama yang sama. Program diatur sebagai kumpulan *package*. Setiap *package* memiliki kumpulan *sub-packagenya* sendiri, yang membantu mencegah bentrok oleh nama *class*

yang sama. Setelah *package* diimport, maka kita bisa menggunakan segala *class* yang ada di dalamnya pada program kita.

Package dalam Java terbagi menjadi dua yaitu, *Built-in-Package* (*package* bawaan dari Java) dan *User-defined-Package* (*package* yang dibuat manual oleh user).

Built-in Package

Built-in package adalah *package* bawaan yang telah disediakan oleh Java. Beberapa contoh *package built-in* yang disediakan Java

- `java.util` menyediakan berbagai *class* untuk menangani berbagai kebutuhan umum (*utility*) seperti *Scanner*, *Date*, *ArrayList*, *HashMap*, *Observer*, *Observable*, dsb.
- `java.io` menyediakan berbagai *class* untuk menangani *input* atau *output* seperti pembacaan *file* sampai penulisan kembali kedalam *disk*.
- `java.sql` menyediakan berbagai *class* untuk menangani koneksi *database*.
- `javax.swing` menyediakan berbagai *class* untuk implementasi *user interface* (UI). Di dalam `javax.swing` terdapat berbagai komponen UI seperti *button*, *input*, *label*, dll.
- `java.lang` berisi *class* dan *interface* yang diperlukan oleh banyak program Java. *Package* ini diimport oleh kompiler ke semua program Java secara otomatis. Jadi tidak perlu mengimpor lagi untuk menggunakan *class* dan *interface* di *package* ini.
- `java.text` berisi *class* dan *interface* yang memperbolehkan program Java.
- `java.net` berisi *class* yang memperbolehkan program untuk berkomunikasi melalui jaringan.

Java memiliki fitur *package* dimana kita dapat mengelompokkan *class* yang memiliki fungsi-fungsi yang saling berkaitan. *Class built in* pada java tentunya telah dikelompokkan dalam *package* tertentu. Agar bisa mengakses *class* dalam *package* tersebut kita perlu mengimpor *package* tersebut kedalam *source code* yang akan kita buat. Kita cukup menulis *statement* `import` lalu dilanjut dengan nama *package* yang ingin digunakan. Berikut merupakan *statement* umum untuk menggunakan suatu *package*. Sebagai contoh, *package* `java.util.Scanner` yang berguna untuk proses *input* nilai dari *user*, kita bisa menggunakan *statement* berikut.

```
import java.util.Scanner;
```

User-Defined Package

User-defined-package adalah *package* yang dibuat dan didefinisikan oleh *user*. Untuk membuat *package*, masukkan perintah *package* sebagai *statement* pertama dalam *source file* Java. *Class* apapun yang dideklarasikan di dalam *file* tersebut akan menjadi milik *package* yang ditentukan. *Statement package* mendefinisikan nama *class* yang disimpan. Jika kita menghilangkan *statement package*, nama *class* akan dimasukkan ke dalam *package default*, yang tidak memiliki nama dan disebut sebagai *un-named package* (*package* tanpa nama). Berikut merupakan contoh *syntax statement user-defined package*.

```
package package_nama;
```

Java menggunakan direktori *file* untuk menyimpan *package*. Perlu diingat bahwa hal ini penting dan nama direktori harus sama persis dengan nama *package*. Lebih dari satu *file* dapat menyertakan *statement package* yang sama. Berikut merupakan contoh dari *user-defined-package*.

Program (Informatika.java)

```
package Gunadarma; //deklarasi package
class Informatika //deklarasi class
{
    public static void main(String[] args)
    {
        System.out.println("Belajar Bahasa Java.");
    }
}
```

Output Program (Informatika.java)

```
Belajar Bahasa Java.
```

Sub-Package

Sub-package adalah *package* yang dibuat di dalam *package* lain. Sebuah *package* dapat terdiri dari *sub-package* (*package* didalam *package*). Penggabungan antara *package* dengan *sub-packagenya* ditulis dengan menggunakan tanda titik (*dot*) sebagai pemisah. Berikut ini contoh *syntax* dari *statement sub-package*.

```
package namaPackage.subPackage;
```

Contoh:

- *Package* `Gunadarma.fakultas` artinya `fakultas` adalah *sub-package* dari `Gunadarma` digunakan untuk mengelompokkan *class* dari fakultas.
- *Package* `Gunadarma.jurusan` artinya `jurusan` adalah *sub-package* dari `Gunadarma` digunakan untuk mengelompokkan *class* dari jurusan.

1.4.2 Library

Library adalah sekumpulan *package* atau koleksi *class* yang telah disediakan oleh Java. Untuk menggunakan *library* dalam java kita menggunakan *syntax* `import`. *Syntax* `import` digunakan untuk memasukan *method* dari *class* atau *library* yang lain, sehingga *method* tersebut dapat digunakan pada *class* yang memanggilnya. Fungsi ini harus diletakan pada baris awal program.

Library dibagi ke dalam *package* dan *class*, yang berarti kita dapat mengimpor satu *class* (bersama dengan *method* dan *attributenya*) atau seluruh *package* yang berisi semua *class* yang termasuk dalam *package* yang ditentukan. Untuk menggunakan sebuah *class* atau sebuah *package* dari *library*, kita harus menggunakan *keyword* `import` seperti berikut.

```
import package.nama.Class; // Import sebuah class
import package.nama.*;     // Import semua ini dalam package
```

Tanda `*` pada `import package.nama.*` berartikan program akan mengimpor seluruh *class*, *method*, dan *attribute* yang ada pada `package.nama` ke dalam *class* yang memanggilnya.

1.5 Input and Output

Seperti yang kita ketahui, program komputer terdiri dari tiga komponen utama, yaitu *input*, *proses*, dan *output*. *Input* merupakan nilai yang kita masukkan ke program, lalu proses merupakan langkah untuk mengelola *input* menjadi sesuatu yang berguna. Sedangkan *output* adalah hasil pengolahan. Semua bahasa pemrograman telah menyediakan fungsi-fungsi untuk melakukan *input* dan *output*. Java sendiri sudah menyediakan *class* untuk proses *input* yaitu *class Scanner* dan untuk *output*, Java menyediakan fungsi `print()`, `println()`, dan `printf()`.

1.5.1 Input

Input dengan menggunakan Class Scanner

Scanner merupakan sebuah *class* dalam *package* `java.util` yang digunakan untuk mendapatkan *input* dari tipe data primitif seperti `int`, `double`, `string`, dan lain-lain. Ini adalah cara termudah untuk membaca *input* dalam pemrograman Java. Secara *default*, *class scanner* bekerja dengan cara memecah *input* menjadi token yang dibatasi oleh karakter spasi, hal ini menyediakan banyak metode untuk membaca dan melakukan *parsing* pada berbagai nilai primitif. Adapun langkah-langkah menggunakan *class scanner* adalah sebagai berikut.

1. Melakukan *import class scanner* yang terdapat pada *package* `java.util`. Bentuk penulisannya adalah sebagai berikut.

```
import java.util.Scanner;
```

2. Membuat *object* referensi sebagai media penginputan data dari *keyboard*. Bentuk penulisannya adalah sebagai berikut.

```
Scanner keyboard = new Scanner(System.in);
```

Kode diatas merupakan inisialisasi awal ketika ingin menggunakan *class Scanner* dan kode diatas akan membuat *object* dapat membaca data masukan dari `System.in`. Syntax `new Scanner(System.in);` berfungsi untuk menciptakan sebuah *object* dari jenis *Scanner*. Syntax `Scanner keyboard` menyatakan bahwa *input* dari *keyboard* adalah variabel yang bertipe `Scanner`. Keseluruhan dari `Scanner keyboard = new Scanner(System.in);` berfungsi untuk membuat *object Scanner* dan memberikan referensi ke *input* variabel.

Method pada Class Scanner

Berikut ini adalah *method* yang biasa digunakan pada *class Scanner*.

Method	Keterangan
<code>nextByte</code>	Membaca input dalam tipe <code>byte</code>
<code>nextDouble</code>	Membaca input dalam tipe <code>double</code>
<code>nextFloat</code>	Membaca input dalam tipe <code>float</code>
<code>nextInt</code>	Membaca input dalam tipe <code>int</code>
<code>nextLine</code>	Membaca input dalam tipe <code>String</code>
<code>nextLong</code>	Membaca input dalam tipe <code>Long</code>
<code>nextShort</code>	Membaca input dalam tipe <code>short</code>

Berikut merupakan contoh program untuk menginput data mahasiswa menggunakan *class Scanner*.

Program (DataMahasiswa.java)

```
package mahasiswa;

// Mengimpor Scanner ke program
import java.util.Scanner;

public class DataMahasiswa
{
    public static void main(String[] args)
    {
        // Deklarasi variabel
        String nama, jurusan;
        int usia, npm;

        // Membuat scanner baru
        Scanner keyboard = new Scanner(System.in);

        // Menampilkan output
        System.out.println("### Pendataan Mahasiswa Gunadarma ###");

        // Menampilkan output
        System.out.print("Nama Mahasiswa: ");
        // Menggunakan scanner dan menyimpan masukkan nilai pada variabel nama
        nama = keyboard.nextLine();

        // Menampilkan output
        System.out.print("Jurusan: ");
        // Menggunakan scanner dan menyimpan masukkan nilai pada
        // variabel jurusan
        jurusan = keyboard.nextLine();

        // Menampilkan output
        System.out.print("Usia: ");
        // Menggunakan scanner dan menyimpan masukkan nilai pada variabel usia
        usia = keyboard.nextInt();
    }
}
```



```
// Menampilkan output
System.out.print("Npm: ");
// Menggunakan scanner dan menyimpan masukan nilai pada variabel npm
npm = keyboard.nextInt();

// Menampilkan apa yang sudah simpan di variabel
System.out.println("-----");
System.out.println("Nama Mahasiswa: " + nama);
System.out.println("Jurusan: " + jurusan);
System.out.println("Usia: " + usia + " tahun");
System.out.println("Npm: " + npm);
}
}
```

Output Program (DataMahasiswa.java)

```
### Pendataan Mahasiswa Gunadarma ###
Nama Mahasiswa: Samsul Arif
Jurusan: Teknik Informatika
Usia: 22
Npm: 54416527
```

Dapat dilihat bahwa dari program diatas, *user* diminta untuk menginput Nama Mahasiswa, Jurusan, Usia dan Npm. Lalu setelah pengguna memasukkan semua *input keyboard* dan mengakhiri *input* dengan menekan tombol ENTER, maka program akan menampilkan sesuai yang telah di *input* seperti berikut.

```
-----
Nama Mahasiswa: Samsul Arif
Jurusan: Teknik Informatika
Usia: 22 tahun
Npm: 54416527
```

1.5.2 Output

Dalam Java, untuk menampilkan suatu teks ke *standard output* kita akan menemui 3 jenis *method* yang akan digunakan, yaitu:

- `System.out.print()`
- `System.out.println()`
- `System.out.printf()`

Output menggunakan `print()`

Method `print()` adalah *method* yang berfungsi untuk mencetak data, setelah data tersebut dicetak tidak diikuti dengan perpindahan baris baru. Jadi jika menginginkan fungsi ini memiliki fasilitas untuk berpindah baris, maka terpaksa kita harus mencantumkan `\n` diakhir *syntax*. Terkadang data yang ditampilkan tidak selalu berupa teks. Jika kita ingin menggabung *output* berupa teks dan variabel, maka yang harus kita lakukan adalah menambahkan karakter `+` sebagai penyambung antara teks dan variabel. Berikut merupakan contoh *output* penggabungan teks dan variabel.

```
system.out.print("NPM : "+npm+", Nama : "+nama+'\n');
```

Sebagai contoh, variabel `npm` dan variabel `nama` akan diberi nilai, yaitu `npm= "54417625"` dan `nama="Budi"`. Maka akan menghasilkan *output* sebagai berikut.

```
NPM : 54417625, Nama : Budi
```

Berikut merupakan contoh program untuk *output* suatu nilai menggunakan fungsi `System.out.print()`.

Program (ContohFungsiOutPrint.java)

```
public class ContohFungsiOutPrint
{
    public static void main(String[] args)
    {
        System.out.print("Belajar ");
        System.out.print("Java\n");

        String npm="10506357";
        String nama="Amar Farizky";

        System.out.print("NPM : "+npm+'\n');
        System.out.print("Nama : "+nama+'\n');
        System.out.print("NPM : "+npm+", Nama : "+nama+'\n');
    }
}
```

Output Program (ContohFungsiOutPrint.java)

```
Belajar Java
NPM : 10506357
Nama : Amar Farizky
NPM : 10506357, Nama : Amar Farizky
```

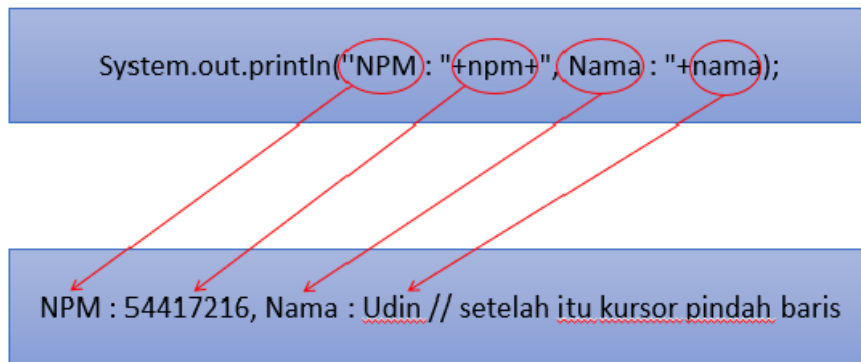
Output menggunakan `println()`

Fungsi *method* `println()` berfungsi untuk mencetak data, setelah data tersebut dicetak diikuti dengan perpindahan baris baru. Sehingga tidak perlu mencantumkan `\n` untuk perpindahan baris baru. Konsep dari *method* `println()` sama dengan `print()`. Dimana jika kita ingin menggabungkan *output* teks dan variabel, maka harus digabung dengan menggunakan tanda `+` seperti contoh berikut.

```
System.out.println("NIM : "+nim+", Nama : "+nama);
```

Sebagai contoh, variabel `nim` dan variabel `nama` akan diberi nilai, yaitu `nim= "54417216"` dan `nama="Udin"`. Maka akan menghasilkan *output* sebagai berikut.

```
NIM : 54417216, Nama : Udin
```



Program (ContohFungsiOutPrintIn.java)

```
public class ContohFungsiOutPrintIn
{
    public static void main(String[] args)
    {
        System.out.println("Belajar ");
        System.out.println("Java");

        String npm="10506357";
        String nama="Amar Farizky";

        System.out.println("NPM : "+npm);
        System.out.println("Nama : "+nama);
        System.out.println("NPM : "+npm+", Nama : "+nama);
    }
}
```

Output Program (ContohFungsiOutPrintIn.java)

```
Belajar
Java
NPM : 10506357
Nama : Amar Farizky
NPM : 10506357, Nama : Amar Farizky
```

Operator Penggabungan String

Operator penggabungan *string* (*string concatenation operator*) digunakan untuk merangkai dua atau lebih *string*. Operator penggabungan *string* menggunakan tanda penjumlahan `+`. Apabila keduanya adalah *string*, maka kedua *string* tersebut langsung dirangkai. Jika salah satunya bukan *string*, maka yang bukan *string* akan dikonversi terlebih dahulu ke dalam *string* dan kemudian baru dirangkakan dengan *string* lainnya. Pada operator penggabungan *string*, konversi ke bentuk *string* dapat dilakukan otomatis oleh Java. Berikut merupakan contoh penggunaan operator penggabungan *string* dengan menggunakan fungsi `println()`.

```
System.out.println("Mari kita belajar pemrograman " + "menggunakan java");
```

Maka akan menghasilkan *Output* seperti berikut.

```
Mari kita belajar pemrograman menggunakan java
```

Output Menggunakan `printf()`

Method `printf()` berfungsi sebagai *output* dengan menentukan penentu format. Itulah sebabnya nama *method* `printf()`, huruf `f` berasal dari singkatan format.

```
System.out.printf("format-string", argumen);
```

Format-string adalah tulisan yang akan tampil di layar monitor. Didalam *format-string*, kita menentukan penentu format yang nantinya akan ditampilkan melalui argumen. Argumen adalah variabel/data yang akan ditampilkan berdasarkan apa yang sudah didaftarkan pada *format-string*.

Format Syntax Specifier

Format *syntax specifier* yang berfungsi untuk menampilkan angka memiliki *syntax* sebagai berikut.

```
%[flag][lebar][.presisi]konversi
```

Penjelasan dari *syntax* di atas adalah sebagai berikut.

- `%` - Setiap format *specifier* harus diawali dengan karakter persen `%`
- `flag` - *flag* sendiri bersifat opsional. *flag* berfungsi untuk membuat nilai diformat dengan berbagai cara.
- `lebar` - lebar berfungsi untuk menentukan lebar minimum dari nilai yang ingin kita format.
- `.presisi` - Jika kita mencetak nilai bertipe *floating point*, maka kita bisa membulatkan nilai tersebut ke suatu presisi. Presisi adalah banyaknya di tempat desimal.
- `konversi` - Semua format *specifier* berakhiran dengan karakter konversi, seperti `f` untuk *floating point* atau `d` untuk integer.

Format *specifier* tidak hanya digunakan untuk memformat angka saja, tetapi dapat digunakan untuk memasukkan suatu nilai dari variabel didalam posisi tertentu di bagian *output*. Sebagai contoh, kita ingin mencetak *output* sebagai berikut.

```
Saya mempunyai 2 motor dan 1 mobil.
```

Berikut merupakan *statement* untuk menghasilkan *output* seperti diatas menggunakan format *specifier*.

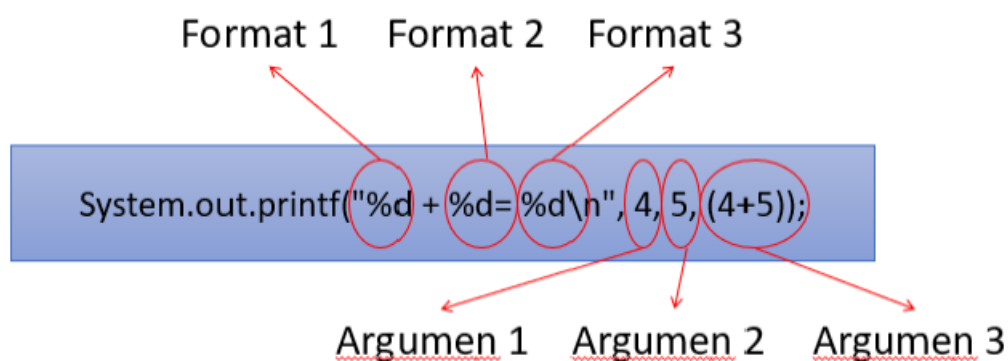
```
int motor = 2;  
int mobil = 1;  
System.out.printf("Saya mempunyai %d motor dan %d mobil.\n", motor, mobil);
```

Dapat dilihat bahwa format *specifier* memudahkan kita untuk memasukkan sebuah nilai ke tengah-tengah String. Format *specifier* yang kita pakai diatas adalah `%d` yang berfungsi sebagai tempat untuk menempatkan nilai yang bertipe data integer. Berikut merupakan daftar penentu format *specifier* bahasa Java.

Format	Untuk menampilkan variabel/data berjenis
%d	Bilangan bulat (integer)
%f	Bilangan pecahan (floating-point)
%c	Karakter 1 digit (char)
%s	Karakter banyak digit (String)
%b	Nilai true/false (boolean)

Berikut contoh *output* menggunakan `printf()`.

```
System.out.printf("%d + %d= %d\n", 4, 5, (4+5));
```



Sehingga *output* yang dihasilkan adalah sebagai berikut.

```
4 + 5= 9
```

Program (ContohFungsiOutPrintf.java)

```
public class ContohFungsiOutPrintf
{
    public static void main(String[] args)
    {
        System.out.printf("%s\n", "Belajar Java");
        System.out.printf("%s %s\n", "Belajar", "Java");
        System.out.printf("%d + %d= %d\n", 4, 5, (4+5));
        System.out.printf("PHI= %f\n", 3.14);
        System.out.printf("index= %c\n", 'A');
        System.out.printf("nilai boolean= %b\n", true);
    }
}
```

Output Program (ContohFungsiOutPrintf.java)

```
Belajar Java
Belajar Java
4 + 5= 9
PHI= 3.140000
index= A
nilai boolean= true
```

1.6 Algoritma

Algoritma adalah urutan langkah logis tertentu untuk memecahkan suatu masalah. Urutan langkah logis yang dimaksud adalah algoritma harus mengikuti suatu urutan tertentu, tidak boleh melompat-lompat. Algoritma mengikuti alur pikiran, sehingga algoritma seseorang dapat juga berbeda dari algoritma orang lain. Algoritma dapat berupa kalimat, gambar, atau tabel tertentu. Sebagai contoh, misal kita ingin mengirim surat kepada kenalan di tempat lain, maka algoritmanya adalah sebagai berikut.

- Menulis surat
- Surat dimasukkan kedalam amplop tertutup
- Amplop ditemplei perangko secukupnya
- Pergi ke kantor pos terdekat untuk mengirimkannya

Dalam bidang komputer, algoritma sangat diperlukan dalam menyelesaikan berbagai masalah pemrograman, terutama dalam komputasi numeris. Tanpa algoritma yang dirancang baik, maka proses pemrograman akan menjadi salah, rusak, atau lambat dan tidak efisien.

Untuk membuat algoritma, kita bisa menggunakan 3 cara berikut yaitu Diagram UML, *Pseudocode* dan Flowchart.

1.6.1 UML

UML merupakan singkatan dari "*Unified Modelling Language*" yaitu suatu metode permodelan secara visual untuk sarana perancangan sistem berorientasi *object*, atau definisi UML yaitu sebagai suatu bahasa yang sudah menjadi standar pada visualisasi, perancangan dan juga pendokumentasian sistem *software*. Saat ini UML sudah menjadi bahasa standar dalam penulisan *blue print software*.

Tujuan atau Fungsi UML





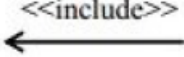
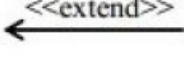
- Dapat memberikan bahasa permodelan visual kepada pengguna dari berbagai macam pemrograman maupun proses rekayasa.
- Dapat menyatukan praktek-praktek terbaik yang ada dalam permodelan.
- Dapat memberikan model yang siap untuk digunakan, merupakan bahasa permodelan visual yang ekspresif untuk mengembangkan sistem dan untuk saling menukar model secara mudah.
- Dapat berguna sebagai *blue print*, sebab sangat lengkap dan detail dalam perancangannya yang nantinya akan diketahui informasi yang detail mengenai *coding* suatu program.

Jenis-jenis Diagram UML dan Contoh Diagram UML

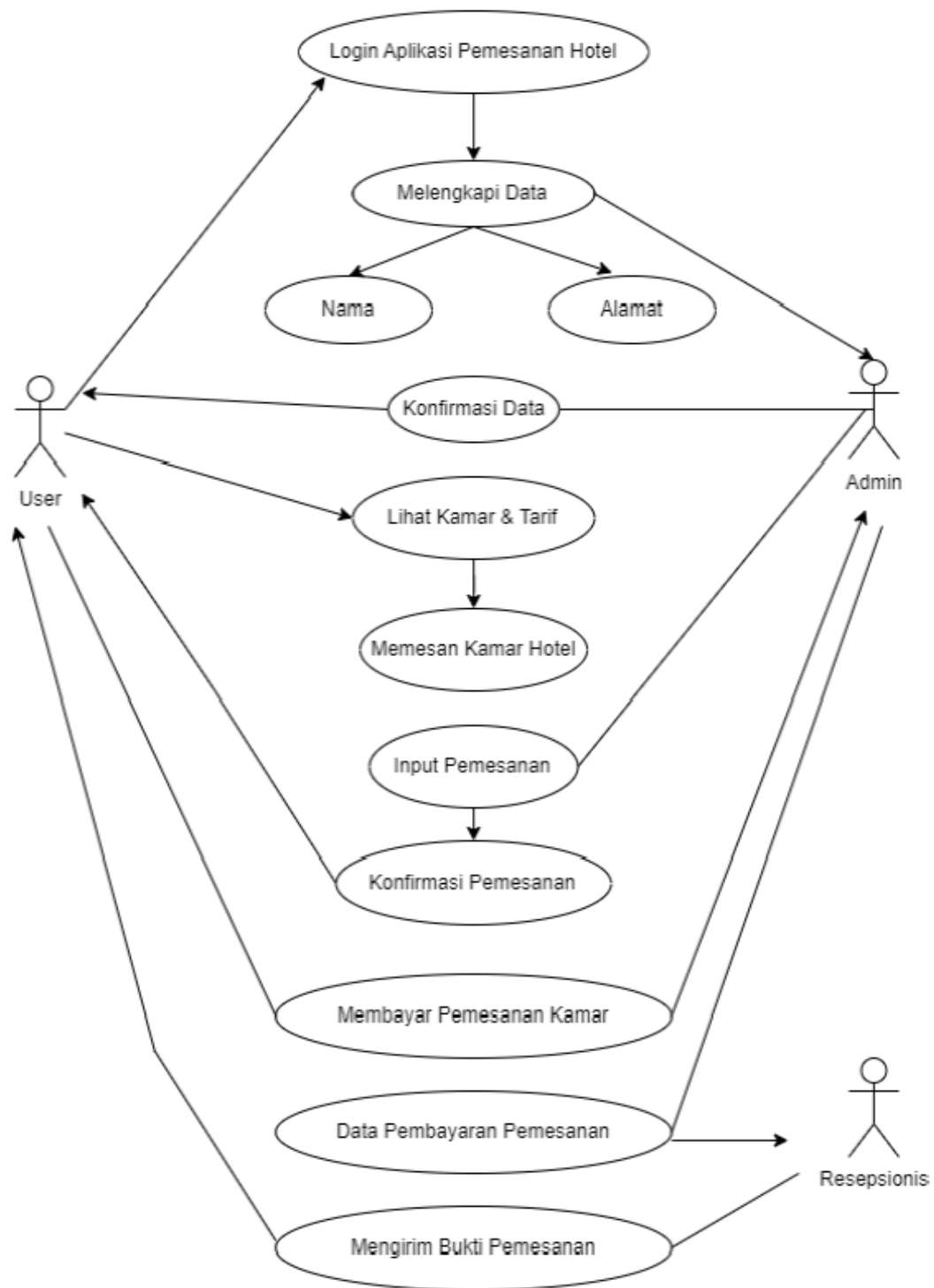
1. Use Case Diagram

Use case diagram adalah salah satu jenis diagram pada UML yang menggambarkan interaksi antara sistem dan aktor, *use case* diagram juga dapat mendeskripsikan tipe interaksi antara si pemakai sistem dengan sistemnya.

Simbol-simbol *Use Case* Diagram adalah sebagai berikut.

Simbol	Keterangan
	Aktor : Mewakili peran orang, sistem yang lain, atau alat ketika berkomunikasi dengan <i>use case</i>
	<i>Use case</i> : Abstraksi dan interaksi antara sistem dan aktor
	<i>Association</i> : Abstraksi dari penghubung antara aktor dengan use case
	<i>Generalisasi</i> : Menunjukkan spesialisasi aktor untuk dapat berpartisipasi dengan use case
	Menunjukkan bahwa suatu use case seluruhnya merupakan fungsionalitas dari use case lainnya
	Menunjukkan bahwa suatu use case merupakan tambahan fungsional dari use case lainnya jika suatu kondisi terpenuhi

Berikut ini adalah contoh *use case* diagram pada pemesanan hotel secara *online*.









Alur pembacaan *use case* diagram pada pemesanan hotel secara *online* adalah sebagai berikut.

- Pelanggan *login* ke aplikasi pemesanan hotel *online*, lalu melengkapi data diri berupa nama, alamat, dll kemudian mengirimkannya ke Admin.
- Admin mengonfirmasi data pelanggan.
- Setelah dikonfirmasi, lalu pelanggan melihat-lihat kamar yang ingin dipesan dan mengecek tarifnya kemudian memesan kamar kepada Admin.
- Admin menginput pemesanan lalu mengonfirmasi kembali pesanan kepada pelanggan.
- Lalu pelanggan membayar kamar yang sudah dipesan kepada Admin.
- Kemudian Admin memberikan data pembayaran kepada Resepsionis.
- Setelah itu, Resepsionis mengirimkan bukti pemesanan kepada Pelanggan.

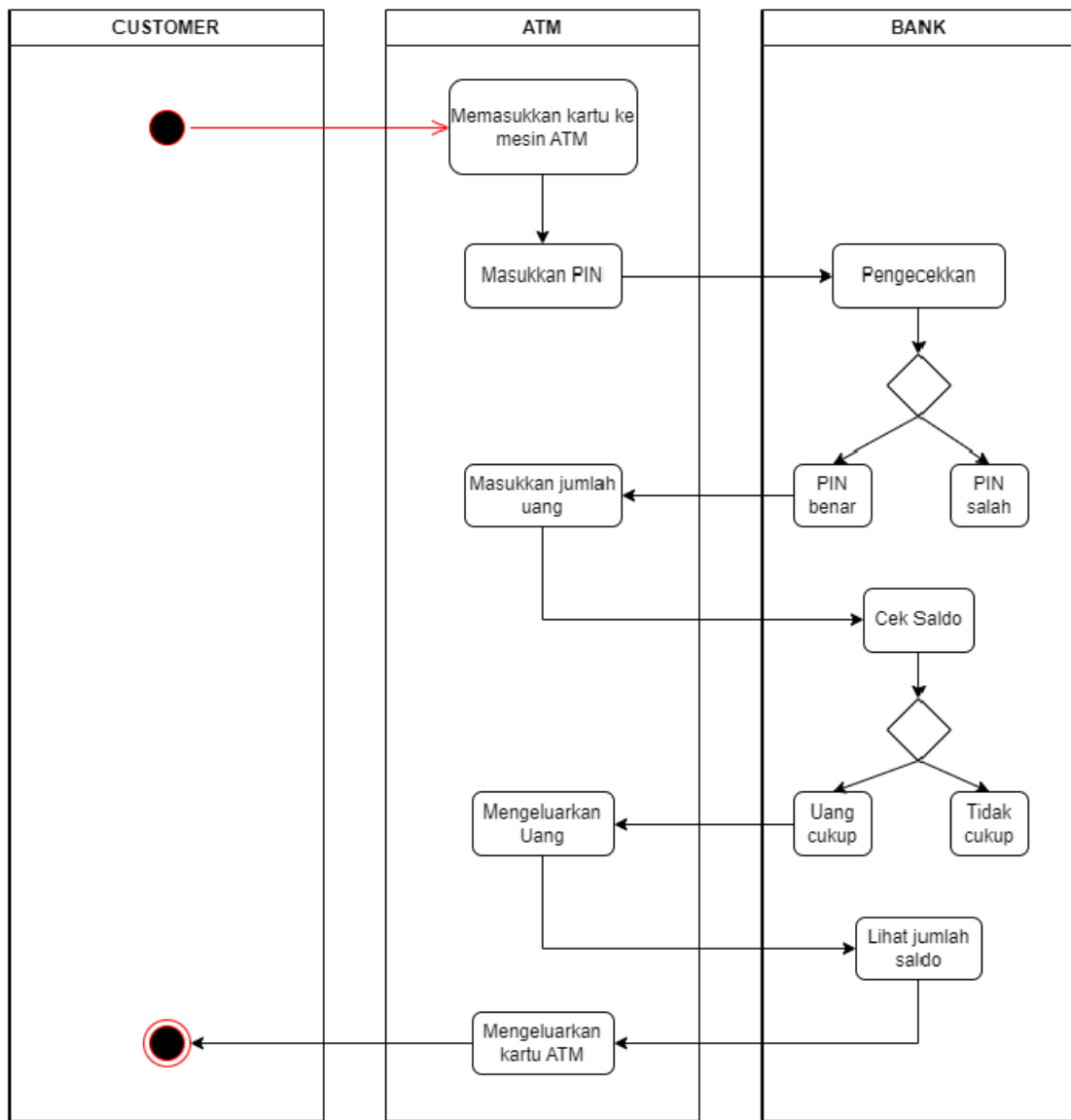
2. Activity Diagram

Activity diagram atau diagram aktivitas yaitu salah satu jenis diagram pada UML yang dapat memodelkan proses-proses apa saja yang terjadi pada sistem.

Simbol-simbol *Activity Diagram* adalah sebagai berikut.

Simbol	Nama	Keterangan
	Status awal	Sebuah diagram aktivitas memiliki sebuah status awal.
	Aktivitas	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja.
	Percabangan / Decision	Percabangan dimana ada pilihan aktivitas yang lebih dari satu.
	Penggabungan / Join	Penggabungan dimana yang mana lebih dari satu aktivitas lalu digabungkan jadi satu.
	Status Akhir	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir
	Swimlane	Swimlane memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi

Berikut ini adalah contoh activity diagram pada penarikan uang di ATM




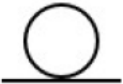




Alur pembacaan *activity* diagram pada penarikan uang di ATM adalah sebagai berikut.

- *Customer* memasukkan kartu ke mesin ATM
- Lalu memasukkan PIN dengan benar
- Kemudian server dari bank akan mengecek PIN benar/salah
- Jika PIN yang dimasukkan benar *Customer* memilih penarikan dan memasukkan jumlah uang
- Dan pihak dari bank akan mengecek saldo
- Jika saldo cukup maka uang dapat keluar dan menampilkan sisa saldo
- Setelah itu kartu keluar dan dapat ditarik kembali

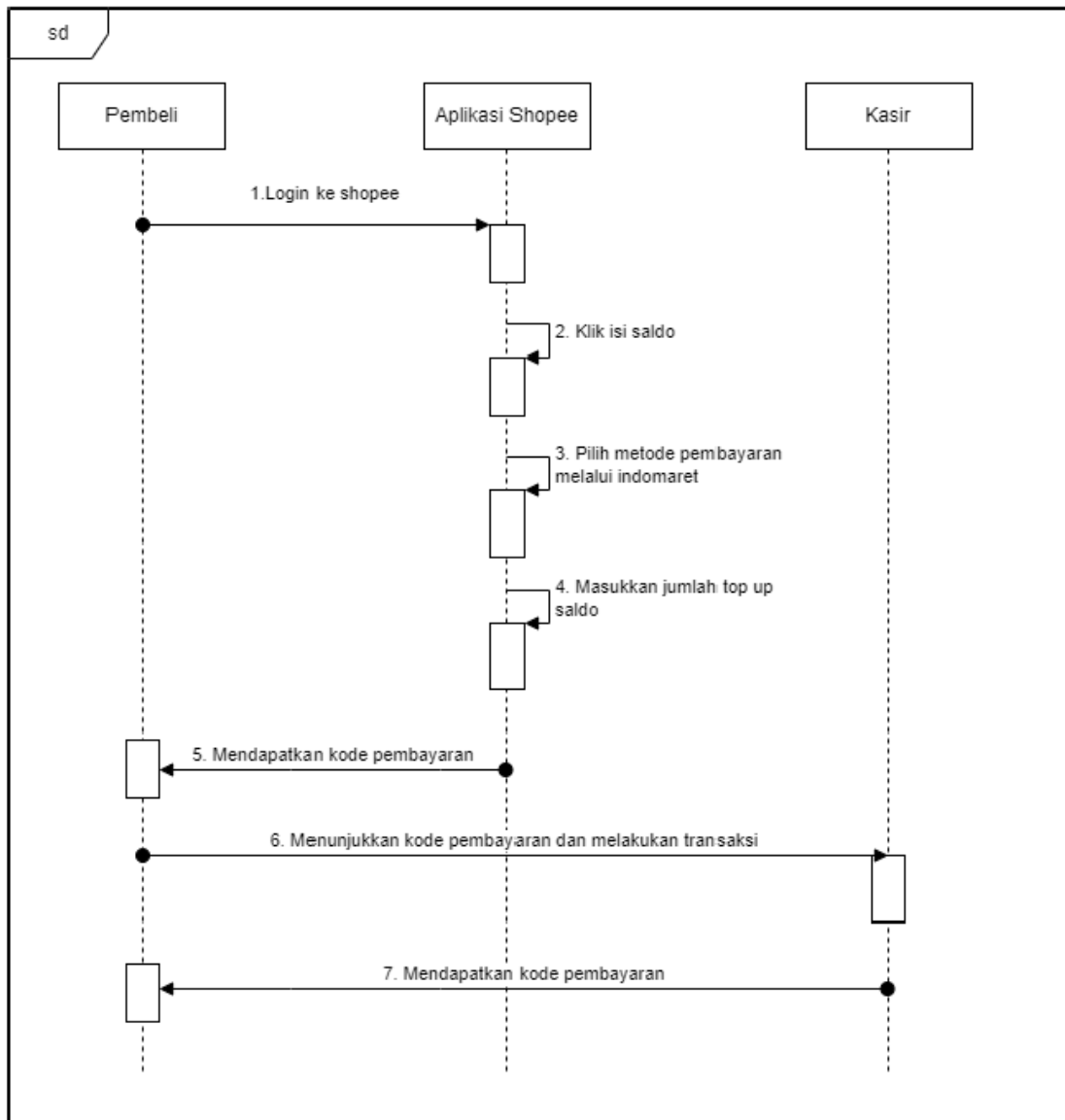
3. Sequence Diagram

Sequence diagram adalah salah satu jenis diagram pada UML yang menjelaskan interaksi *object* yang berdasarkan urutan waktu, *sequence* diagram juga dapat menggambarkan urutan atau tahapan yang harus dilakukan untuk dapat menghasilkan sesuatu seperti pada *use case* diagram.

Simbol-simbol *Sequence* Diagram adalah sebagai berikut.

NO	GAMBAR	NAMA	KETERANGAN
1		<i>Actor</i>	Menggambar orang yang sedang berinteraksi dengan sisitem.
2		<i>Entity Class</i>	Menggambarkan hubungan yang akan dilakukan
3		<i>Boundary Class</i>	Menggambarkan sebuah gambbaran dari foem
4		<i>Control Class</i>	Menggambarkan penghubung antara boundary dengan tabel
5		<i>A focus of Control & A Life Line</i>	Menggambarkan tempat mulai dan berakhirnya massage
6		<i>A massage</i>	Menggambarkan Pengiriman Pesan

Berikut ini adalah contoh *sequence diagram* tentang pengisian saldo *shopee pay* melalui indomaret.



Alur pembacaan *sequence* diagram pada pengisian saldo *shopee pay* melalui indomaret adalah sebagai berikut.








- Pembeli *login* ke aplikasi shopee
- Kemudian klik isi saldo
- Lalu pilih metode pembayaran melalui indomaret
- Setelah itu masukkan jumlah *Top up* saldo yang diinginkan
- Lalu pembeli mendapatkan kode pembayaran
- Kemudian pembeli menunjukkan kode pembayaran dan melakukan transaksi kepada kasir indomaret
- Setelah itu pembeli akan mendapatkan bukti pembayaran dari kasir

4. Class Diagram

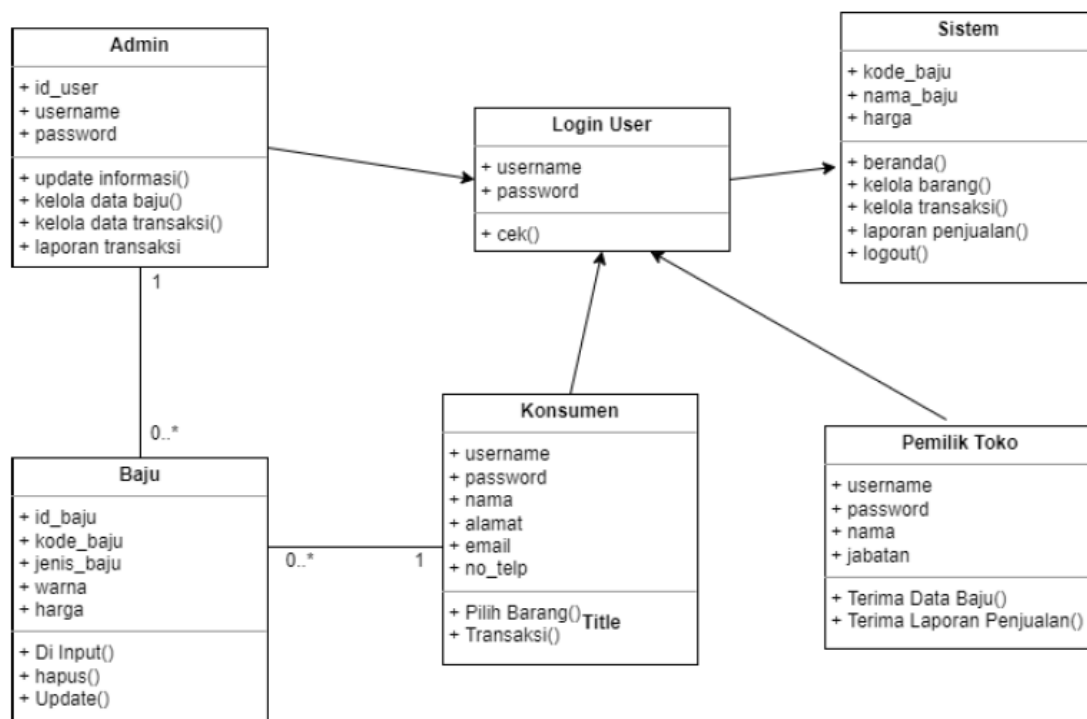
Class diagram adalah salah satu jenis diagram pada UML yang digunakan untuk menampilkan *class* maupun paket-paket yang ada pada suatu sistem yang nantinya akan digunakan. Jadi diagram ini dapat memberikan sebuah gambaran mengenai sistem maupun relasi-relasi yang terdapat pada sistem tersebut.

Simbol-simbol *Class* Diagram adalah sebagai berikut.

SIMBOL CLASS DIAGRAM

NO	GAMBAR	NAMA	KETERANGAN
1		<i>Generalization</i>	Hubungan dimana objek anak (<i>descendent</i>) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (<i>ancestor</i>).
2		<i>Nary Association</i>	Upaya untuk menghindari asosiasi dengan lebih dari 2 objek.
3		<i>Class</i>	Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
4		<i>Collaboration</i>	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu actor
5		<i>Realization</i>	Operasi yang benar-benar dilakukan oleh suatu objek.
6		<i>Dependency</i>	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri (<i>independent</i>) akan memengaruhi elemen yang bergantung padanya elemen yang tidak mandiri
7		<i>Association</i>	Apa yang menghubungkan antara objek satu dengan objek lainnya

Berikut ini adalah contoh *class diagram* pada pembelian melalui aplikasi *shopee*.



Alur pembacaan *class diagram* pada pembelian melalui aplikasi *shopee* adalah sebagai berikut.

- Terdapat 6 *class* diantaranya yaitu Admin, Baju, Konsumen, Pemilik Toko, Sistem, dan Login User
- Kelas *Login User* sebagai kelas penghubung antara Admin, Baju, Konsumen, dan Pemilik Toko
- Kelas *Login User* terdiri dari 2 atribut dan 1 operasi

- Kelas Konsumen terdiri dari 6 atribut dan 2 operasi
- Kelas Admin terdiri dari 3 atribut dan 4 operasi
- Kelas Baju terdiri dari 5 atribut dan 3 operasi
- Kelas Pemilik Toko terdiri dari 4 atribut dan 2 operasi
- Kelas Sistem terdiri dari 3 atribut dan 5 operasi

1.6.2 Pseudocode

Pseudocode merupakan sebuah kode yang digunakan untuk menulis sebuah algoritma, bahasa yang dipakai hampir menyerupai bahasa pemrograman tingkat tinggi. *Pseudocode* berisi langkah-langkah untuk menyelesaikan suatu permasalahan dan penulisannya bebas tidak terikat dengan bahasa pemrograman manapun. Adapun ciri-ciri dari pseudocode adalah sebagai berikut.

- Menggunakan pola bahasa Inggris yang sederhana
- Tidak memiliki standar aturan tertentu dalam penulisannya
- *Pseudocode* menggunakan simbol atau *syntax* dari suatu program, seperti \leftarrow , $<$, $>$, $<=$, $>=$, $=$ dan sebagainya.
- Notasi *pseudocode* bisa digunakan untuk bahasa pemrograman
- Tidak menggunakan diagram melainkan ditulis dalam urutan suatu kejadian atau permasalahan
- *Pseudocode* berisi langkah-langkah untuk menyelesaikan sebuah masalah (seperti halnya algoritma), akan tetapi bentuk masalahnya sedikit berbeda dari algoritma
- Sering digunakan para pengguna untuk menuliskan suatu algoritma dari suatu permasalahan

Perbedaan antara Algoritma dan Pseudocode

Algoritma	Pseudocode
Masukkan panjang	Input panjang
Masukkan lebar	Input lebar
Nilai luas adalah panjang x lebar	Luas =panjang x lebar
Tampilkan luas	Print luas

Untuk menuliskan algoritma *pseudocode*, dibutuhkan tiga struktur dasar, yaitu:

- Judul
- Deskripsi
- Implementasi

Judul

Judul yang dipakai dalam *pseudocode* adalah judul algoritma yang akan dipakai atau judul yang ingin dibuat oleh penulis. Contohnya jika kita ingin membuat sebuah program untuk menentukan keliling persegi, maka judul akan ditulis seperti berikut.

Program <Nama_Program> maka ditulis,

- Program Menentukan_Keliling_Persegi
- Program Menghitung_luas_persegi_panjang

Deskripsi

Bagian ini berisi deklarasi dari keterangan algoritma yang akan dibuat, yaitu keterangan variabel (var) atau konstanta yang digunakan untuk menghitung suatu rumus tertentu. Variabel adalah wadah dari data yang akan digunakan. Contoh penulisannya adalah sebagai berikut.

Deskripsi <NamaVariabel: <tipe_data>; maka ditulis

- Deskripsi
Var sisi, keliling : integer;
- Deskripsi
var panjang, lebar, luas : integer;

Implementasi

Bagian ini berisi proses atau langkah-langkah yang akan dilakukan algoritma atau inti dari algoritma itu sendiri. Maksudnya adalah pengguna harus menuliskan besaran angka pada masing-masing variabel yang akan dihitung dan sebagainya. Contoh penulisannya adalah sebagai berikut.

Implementasi (berisi inti dari Algoritma tersebut); maka ditulis

- Implementasi
Input (sisi);
keliling = sisi*4;
Print (keliling);
- Implementasi
Input (panjang);
Input (lebar);
luas = panjang*lebar;
Print (luas);

Berikut merupakan contoh *pseudocode*.

Algoritma *pseudocode* untuk menentukan keliling persegi

```
Judul: Program Menentukan_Keliling_Persegi
Deklarasi
var sisi,keliling: integer;
Implementasi
Input(sisi);
keliling = sisi*4;
Print(keliling);
```

Algoritma *pseudocode* untuk menentukan luas persegi panjang

```
Judul: Program Menghitung_luas_persegi_panjang
Deklarasi
var panjang, lebar, luas : integer;
Implementasi
Input(panjang);
Input(lebar);
luas = panjang*lebar;
Print(luas);
```

1.6.3 Flowchart

Flowchart merupakan bentuk algoritma yang menggambarkan suatu sistem dengan menggunakan simbol-simbol serta menjelaskan suatu urutan serta hubungan proses didalam sistem. *Flowchart* sering digunakan sebagai pedoman untuk menjalankan operasional dan juga dokumentasi. Ada banyak jenis *flowchart* dan salah satunya adalah *flowchart* program. Terdapat beberapa *flowchart* program yang sering digunakan diantaranya adalah:



Oval (Simbol Terminal)

Menunjukkan permulaan (start) atau akhir (stop) dari suatu proses



Jajar Genjang (Simbol Input/Output)

Menunjukkan proses input/output yang terjadi



Persegi Panjang (Simbol Proses)

Digunakan untuk menunjukkan kegiatan yang dilakukan oleh komputer



Belah Ketupat (Simbol Keputusan)

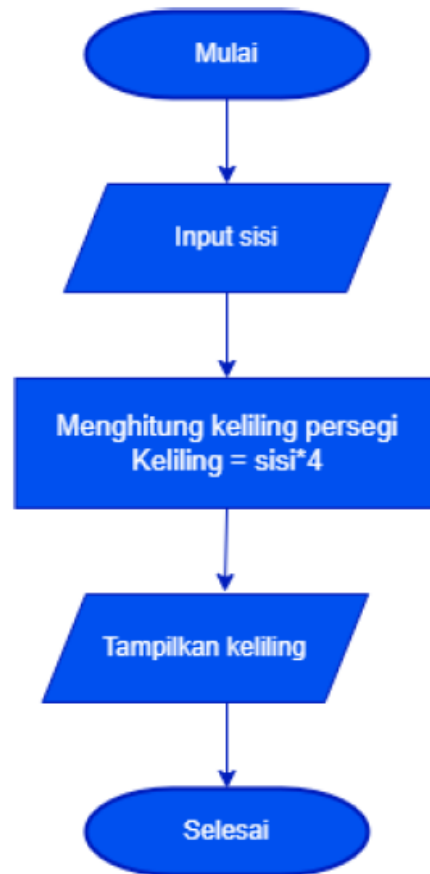
Digunakan untuk memilih keputusan berdasarkan kondisi yang ada



Panah (Simbol Arus)

Digunakan untuk menghubungkan satu simbol dengan simbol lainnya. Simbol ini berfungsi untuk menunjukkan garis alir dari proses

Contoh *flowchart* dari program menghitung keliling persegi adalah sebagai berikut.



Contoh *flowchart* dari program menghitung luas persegi panjang



REFERENSI

- [1] Carol Britton & Jill Doake. A Student Guide to Object Oriented Development. Elsevier ButterworthHeinemann
- [2] Gaddis, Tony. 2016. *Starting Out with Java: From Control Structures through Objects (6th Edition)*. Boston: Pearson.
- [3] Harry .H. Chaudhary. Teach Yourself Programming With Java in 24 Days. Programmers Mind